OHIO SCIENTIFIC

EXTENDED MACHINE LANGUAGE MONITOR

USER'S MANUAL

Extended Machine Language Monitor

Table of Contents

Introduction

The 6500 Extended Monitor is an extensive machine code debugging aid. It includes the following commands for

- memory display and modification

    memory display and change
    memory dump
    memory fill
    memory move
    memory relocate

- program debugging

    disassembly
    search for byte string
    search for character string
    breakpoint installation and control
    processor register display and change
    program execution

- audio cassette input/output

    load
    save
    view

- hexidecimal arithmetic

    calculate
    display overflow/remainder

Loading the Extended Monitor

The method for loading the Extended Monitor depends upon which version you are using. Refer to the appendix appropriate to your system.

After the Extended Monitor has been loaded its prompter, a colon (:) is displayed. This is the Extended Monitor's command mode.

Extended Monitor Commands

Each of the Extended Monitor commands is listed below. Any of these commands may be entered whenever you are in the command mode as indicated by the colon prompter. Many of the commands also have subcommands which can be entered only after the primary command has been entered. If an invalid command is entered, a "?" will be printed.

In the command descriptions below, all addresses and data values are hexidecimal and the following abbreviations or special characters are used:

	Meaning
<LF>	the line feed key on the keyboard
<CR>	the carriage return (or return) key on the keyboard
↑	an up arrow character. May be a ↑, ^ or a shift/N on some keyboards
@	a commercial-at character. May be a shift/P on some keyboards

Memory Display and Modification Commands

@aaaa displays the address and contents of the location aaaa. New contents may or may not then be entered (two hex digits) followed by one of the following:

- <LF> displays the next location
- ↑ displays the previous location
- " prints the contents of the location as an ASCII or graphic character
- <CR> exits to the Extended Monitor command mode

Dffff,tttt dumps the contents of memory locations ffff through tttt-1.

Fffff,tttt=dd fills memory locations ffff through tttt-1 with the value dd.

Maaaa=ffff,tttt moves the contents of memory from locations ffff through tttt-1 to the memory starting at location aaaa.

Note: The distance of an upward move must be greater than the length of the move or data in the original locations will be overwritten (aaaa>=tttt or aaaa<ffff).

Raaaa=ffff,tttt — relocates (moves) the contents of memory from locations ffff through tttt-1 to the memory starting at location aaaa and appropriately adjusts all three-byte 6500 instruction operand addresses that refer to locations within the range of the move. (Adds (aaaa-ffff) to each operand address that is >=ffff and <=tttt-1.)

Note: The distance of an upward move must be greater than the length of the move or data in the original locations will be overwritten (aaaa>=tttt or aaaa<ffff).

Program Debugging Commands

/Qffff — disassembles 6500 machine code into 6500 mnemonic code from memory location ffff up. Disassembly continues for a total of 24 lines - a maximum of 72 bytes. At completion awaits:

<LF> disassembles the next 24 lines

<CR> exits to the Extended Monitor command mode

Ndd...dd>ffff,tttt — searches the contents of memory locations ffff through tttt-1 for the string of 1 to 8 data bytes dd...dd.

Wc...c>ffff,tttt — searches the contents of memory locations ffff through tttt-1 for the string of 1 to 8 ASCII characters c...c.

Bn,aaaa — installs breakpoint n (n = 1-8) at address aaaa. The contents of location aaaa is saved and may be restored with the En command. If breakpoint n had previously been assigned it is first restored. When a breakpoint is "hit" during program execution it is also automatically restored. (See Using Breakpoints for Program Debugging)

En — eliminates breakpoint n (n = 1-8) and restores the original contents of the location where it was located.

Gaaaa — goes (transfers program control) to address aaaa.

T — prints a table of breakpoint addresses for each breakpoint 1 through 8. An address of FFFF indicates an unassigned breakpoint.

C — continues program execution from the location of the last breakpoint. This command must only be used after a breakpoint has been "hit".

I — prints the address of the last breakpoint "hit" and the contents of the A, X, Y, processor status (P) and stackpointer (K) registers as they existed at that breakpoint.

A, X, Y, P, K — these five commands print the contents the associated register. New contents may or may not then be entered (two hex digits) followed by one of the following:

" — prints the contents of the register as an ASCII or graphic character

<CR> — exits to the Extended Monitor command mode

Audio Cassette Commands

Sffff,tttt — saves the contents of memory locations ffff through tttt-1 by writing them to the cassette port (as well as the terminal) in checksum format. This function may be terminated by typing "L" and a space.

L — loads into memory the data read from the cassette port in checksum format. If a checksum error is detected, "ERR" is printed. To recover, stop the cassette machine, rewind the tape a short distance and restart playing it. Type an "L" to restart the loading. The LOAD command can be exited at any time by typing a space.

V — view the data read from the cassette port in checksum format. Same as Load, above, but displays the data without modifying memory.

The checksum format is as follows for each "record" of data:

;18aaaadd...ddcccc

where:

- ; — is the start of record character
- 18 — is the hexidecimal number of data bytes in the record (24 decimal)
- aaaa — is the address of the first data byte in the record
- dd...dd — are the 24 data bytes
- cccc — is the checksum - a sum modulo 65536_{10} of all bytes in the record after the start of record character

Each record is followed by a carriage return, line feed and nulls.

Additional Commands

Hxxxx,yyyy+ — calculates the sum of hexidecimal values xxxx and yyyy and prints the result. The following operators are available in addition to +:

- difference, xxxx-yyyy

* product, xxxx*yyyy

/ quotient, xxxx/yyyy

O prints the overflow or remainder from the last multiplication or division performed with the H command.

Data Block Moves vs. Program Relocation

The Move command moves the contents of memory without changing any of the moved data. The Relocate command also moves the contents of memory, but it also adjusts the operands of moved 6500 instructions to reflect their new location. For example, consider the following portion of a program residing at Ø3ØØ to Ø38Ø.

```
Ø3ØØ   A52Ø      LDA $2Ø
Ø3Ø2   2Ø6EØ3     JSR $Ø36E
          ⋮
Ø36E   8DC6DØ     STA $DØC6
          ⋮
```

If the command "MØ5ØØ=Ø3ØØ,Ø38Ø" was executed, the code would be moved, unchanged, to Ø5ØØ and would appear there unchanged. This code at Ø5ØØ would only "work" as long as the subroutine at Ø36E remained.

If, on the other hand, the command "RØ5ØØ=Ø3ØØ,Ø38Ø" were used to make the move, the moved instructions would be relocated and would appear as:

```
Ø5ØØ   A52Ø      LDA $2Ø
Ø5Ø2   2Ø6EØ5     JSR $Ø56E
          ⋮
Ø56E   8DC6DØ     STA $DØC6
          ⋮
```

Note, that the operand of the JSR at Ø5Ø2 was changed to reflect the programs new location. However, the operand of the STA at Ø56E was not changed since it fell outside the range of the original program (Ø3ØØ - Ø38Ø). Specifically, the algorithm for changing operands is:

Each three-byte instruction operand which specifies an address within the originating range of the move is adjusted by adding to

it the difference between its final address and its original address, i.e.:

final operand = original operand + final address - original address

Thus, the operand of the JSR instruction in the above example was changed as follows:

Ø56E = Ø36E + Ø5Ø2 - Ø3Ø2

Note, that not all programs are directly relocatable. Any program with data embedded between instructions may not relocate properly because the relocation routine is likely to misread some data as 6500 opcodes. One possible remedy for this situation is to fill all data bytes and tables within the program with a non-relocatable opcode such as ØØ, then relocate the program, then restore the data. Programs can also be unrelocatable due to the use of addresses in immediate operands and addresses in data. However, it is possible to code programs so that they are directly relocatable, if desired, by refraining from the use of relocatable addresses in data and immediate instructions and by locating all data after all instructions in a program. Then a single relocate command for the instructions and a single move command for the data will relocate a complete program.

Using Breakpoints for Program Debugging

As the name implies, a breakpoint is a point where the execution of a running program may be "broken" or interrupted. Using the Extended Monitor, up to eight breakpoints may be placed into a program. When the program is run (executed) and a breakpoint is encountered, the Extended Monitor is re-entered and prints the following to document the breakpoint:

```
Bn@aaaa
A/aa X/xx Y/yy P/pp K/kk
```

where:

n	is the breakpoint number 1-8
aaaa	is the location where the breakpoint was encountered
aa	is the contents of the accumulator
xx	is the contents of the X index register
yy	is the contents of the Y index register
pp	is the contents of the processor status word
kk	is the contents of the stackpointer

To illustrate the use of a breakpoint, consider the following portion of a program:

```
Ø35Ø   BØØ3     BCS $Ø355
Ø352   4C8ØØ3   JMP $Ø38Ø
Ø355   2ØØØØ5   JSR $Ø5ØØ
```

If a breakpoint were installed at Ø35Ø (by the command, "B1,Ø35Ø"), the opcode BØ would be saved and a BRK opcode, ØØ, would be installed in its place. Then, when the program is executed and the breakpoint at location Ø35Ø is "hit", the Extended Monitor would print, for example:

```
B1@Ø35Ø
A/41 X/ØØ Y/13 P/35 K/FC
```

and breakpoint 1 would automatically be eliminated, i.e., the BRK opcode would be removed from location Ø35Ø and the original opcode, BØ, reinstalled. Upon

examination of the status register (P), we find that the carry flag is set so the program will take the BCS branch to Ø355. If you would prefer to fall through the BCS to the JMP instruction, you could simply reset the carry flag in the processor status register by typing "P" then "34". Then, when the continue command ("C") is given, the BCS branch will not be taken.

Note: Extended Monitor temporaries are stored in page zero locations CØ through FF. Care should be taken not to alter these locations with the program being debugged.

If the Extended Monitor is entered by a BRK that was not installed as a breakpoint, a "?" will be printed. The location of the BRK may be determined with the "I" command.

Extended Monitor Command Summary

Command	Function	Subcommands (<CR> always returns to ":")
@aaaa	display contents of aaaa	dd change aaaa "display as character <LF> display next locatior ↑ display previous locatic
A, X, Y, P or K	display A, X, Y, P or K from last break A, X, Y - processor register P - processor status K - stackpointer	dd change register " display as character
Bn,aaaa	enter breakpoint n at aaaa	(n = 1-8)
C	continue from last breakpoint	
Dffff,tttt	dump ffff through tttt-1	
En	eliminate breakpoint n	
Fffff,tttt=dd	fill ffff through tttt-1 with dd	
Gaaaa	go to aaaa	
Hxxxx,yyyy+	display xxxx+yyyy	(also -, *, /)
I	display location of last breakpoint	
L	load memory from cassette	space key returns to ":"
Maaaa=ffff,tttt	move ffff through tttt-1 to aaaa	
Ndd...dd>ffff,tttt	search ffff through tttt-1 for dd...dd	(dd...dd is 1-8 bytes)
O	display overflow/remainder from last H command	
Qaaaa	disassemble from aaaa	<LF> continue disassembly
Raaaa=ffff,tttt	relocate ffff through tttt-1 to aaaa	
Sffff,tttt	save ffff through tttt-1 to cassette	
T	display breakpoint table	
V	view from cassette	space key returns to ":"
Wc...c>ffff,tttt	search ffff through tttt-1 for c...c	(c...c is 1-8 characters)

Appendix A

OS-65D V3.0 Version

of the

Extended Monitor

In OS-65D V3.0, the Extended Monitor is loaded from disk and initiated by typing EM after the A* prompter in the DOS kernel command mode. Whenever exiting to the DOS, you can return to the Extended Monitor as long as it is loaded by typing RETURN EM.

This version of the Extended Monitor occupies memory from 17ØØ through 1FFF and uses page zero locations CØ through FF.

Appendix B

C2P - 1P Cassette Version

of the

Extended Monitor

This version of the Extended Monitor is supplied on an auto-load cassette tape. The following procedure may be used to load the Extended Monitor from tape:

Loading the Extended Monitor

1. Apply power to your personal computer then reset it by pressing the break key. Load the cassette, label up, into the cassette machine and turn the cassette machine on with the volume at about mid-range.

2. Type "ML".

   The M initiates the 65 VP monitor and the L starts the auto-load. In a few seconds the four zeros in the upper left portion of the video monitor should change to an incrementing address value with a rapidly changing data field. The value of the address is dependent on which auto-load cassette is being read. At this time a checksum loader is being read into memory in 65VP format. Upon completion (no more than 30 seconds), the checksum loader will load the rest of the cassette. The Extended Monitor comes up with the prompter ":".

Should a checksum error occur, the following message is printed:

```
OBJECT LOAD CHECKSUM ERR
REWIND PAST ERR - TYPE G TO RESTART
```

If a checksum error consistently happens at the same location, the cassette is probably bad - contact your OSI dealer concerning replacement. However, should checksum errors occur randomly at various locations, it is most likely that there is a problem with the cassette machine or the connection to the computer. Check for broken or frayed connections. Make sure that the play-back head and pressure roller/capstan assembly is clean. With a minimal amount of care, no problems with auto-load cassettes should be encountered.

This version of the Extended Monitor contains one additional instruction for dumping the contents of memory on the 24 character 1P video screen:

Command	Function	Subcommand
Zaaaa	dumps 8 bytes from aaaa	<LF> dumps next 8 bytes

This version occupies memory from Ø8ØØ through ØFFF and uses page zero locations DØ through FF. The checksum loader used to load the Extended Monitor occupies locations Ø7ØØ through Ø7EF.

This version of the Extended Monitor is normally entered at Ø8ØØ. This causes the stackpointer to be set to Ø1FF and the breakpoint registers to be initialized. Under certain circumstances, it may be desirable to re-enter the Extended Monitor without this initialization. This may be done by entering it at Ø81F.

There are two free command letters - J and U, that can be utilized by inserting the address of a command routine at Ø974 for J or Ø98A for U. The machine language command routine must end with an RTS instruction.